Formal Safety Certification of Aerospace Software

Ewen Denney and Bernd Fischer

{edenney,fisch}@email.arc.nasa.gov

USRA/RIACS, NASA Ames Research Center, Moffett Field, CA 94035, USA

In principle, formal methods offer many advantages for aerospace software development: they can help to achieve ultra-high reliability, and they allow external scrutiny of the reliability claims. However, despite years of research and many advances in specification, semantics, and logic, they are not much used in practice. Three major shortcomings are usually identified as the cause of this. First, their application is expensive because the methods are labor- and knowledge-intensive. Second, they are difficult to scale up to complex systems because they are based on deep mathematical insights about the behavior of the systems (i.e., they rely on the "heroic proof"). Third, the proofs can be difficult to interpret, and typically stand in isolation from the original code.

In this paper, we describe a tool for formally demonstrating the safety of aerospace software, that largely circumvents these problems. We focus on safety because it has been observed[1] that safety violations constitute the majority of the errors found in this domain. In our approach, different aspects of software safety can be formalized as different *safety policies* in Hoare logic, which are then used by a verification condition generator along with the code and logical annotations in order to derive safety conditions; these are proven using an automated theorem prover. Our certification system is currently integrated into a model-based code generation toolset that generates the annotations together with the code. However, this *automated formal certification* technology is not exclusively constrained to generated code and could, in principle, also be applied to legacy code.

Our approach circumvents the historical problems with formal methods by increasing the degree of automation on all levels. The restriction to safety policies (as opposed to arbitrary functional behavior) results in simpler proof problems that can generally solved by fully automatic theorem provers.[2] An automated linking mechanism between the safety conditions and the code provides the traceability mandated by process standards such as DO-178B.[3] An automated explanation mechanism uses semantic markup added by the verification condition generator to produce natural-language explanations of the safety conditions and thus supports their interpretation in relation to the code. Figure 1 shows an automatically generated *certification browser* that lets users inspect the (generated) code along with the safety conditions (including textual explanations), and uses hyperlinks to automate tracing between the two levels. The interface also provides some limited control over the certification process itself.

Our long-term goal is a seamless integration of certification, code generation, and manual coding that results in a "certified pipeline" in which specifications are automatically transformed into executable code, together with the supporting artifacts necessary for achieving and demonstrating the high levels of assurance needed in the aerospace domain.

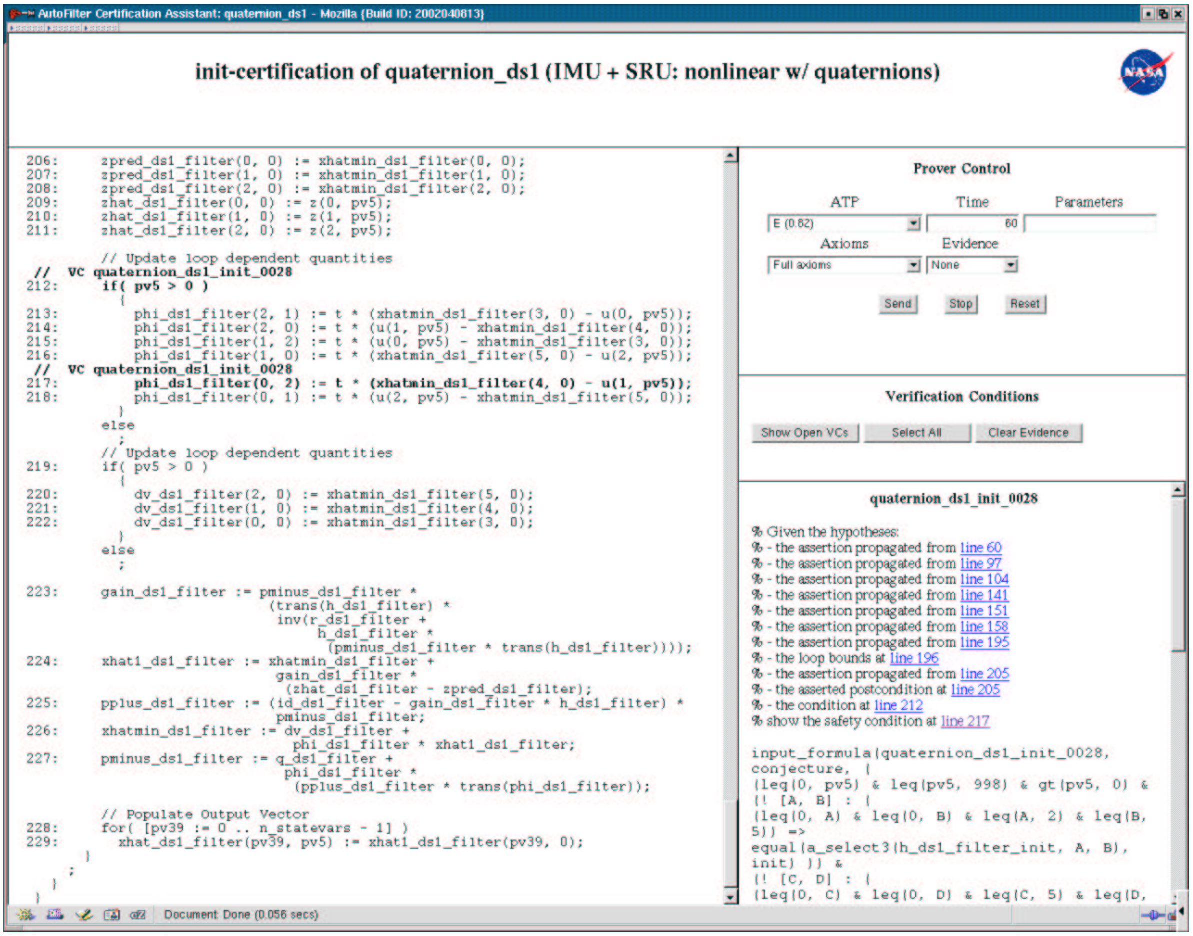

Figure 1. Certification browser

References

[1]Kandt, R., "Software Defect Avoidance and Detection: Practices and Techniques," Tech. rep., JPL, 2003, Document D-24993.

[2]Denney, E., Fischer, B., and Schumann, J., "Using Automated Theorem Provers to Certify Auto-Generated Aerospace Software," *Proceedings of the 2nd International Joint Conference on Automated Reasoning (IJCAR'04)*, Vol. 3097 of *Lecture Notes in Artificial Intelligence*, Cork, Ireland, 2004, pp. 198–212.

[3]RTCA Special Committee 167, "Software Considerations in Airborne Systems and Equipment Certification," Tech. rep., RTCA, Inc., Dec. 1992.